Introducing Haskell

from: COS 441 Slides 3B
by David Walker, Princeton
INDUCTIVE PROOFS
ABOUT HASKELL PROGRAMS
Recall: Proofs by simple calculation

• Some proofs are very easy and can be done by:
  – unfolding definitions
  – using lemmas or facts we already know
  – folding definitions back up

• Eg:

**Definition:**
easy x y z = x * (y + z)

given this

**Theorem:** easy a b c == easy a c b

**Proof:**

easy a b c

= a * (b + c) \hspace{1cm} \text{(by unfold)}

= a * (c + b) \hspace{1cm} \text{(by commutativity of add)}

= easy a c b \hspace{1cm} \text{(by fold)}
**Theorem:** For all finite Haskell lists \( xs \) and \( ys \),
\[
\text{length}(xs ++ ys) = \text{length} \ xs + \text{length} \ ys
\]

**Proof attempt:**

\[
\begin{align*}
[\ ] ++ ys & = ys \\
(x:xs) ++ ys & = x:(xs ++ ys)
\end{align*}
\]

\[
\begin{align*}
\text{length} [ \ ] & = 0 \\
\text{length} (x:xs) & = 1 + \text{length} \ xs
\end{align*}
\]
**Theorem:** For all finite Haskell lists \( xs \) and \( ys \),
\[
\text{length}(xs ++ ys) = \text{length} \, xs + \text{length} \, ys
\]

**Proof attempt:**
case: \( xs = [ ] \)

\[
[ ] ++ ys = ys
\]
\[
(x:xs) ++ ys = x:(xs ++ ys)
\]

case: \( xs = x:xs' \)

\[
\text{length} \, [ ] = 0
\]
\[
\text{length} \, (x:xs) = 1 + \text{length} \, xs
\]
**Theorem:** For all finite Haskell lists \( xs \) and \( ys \),

\[
\text{length}(xs \, ++ \, ys) = \text{length} \, xs \, + \, \text{length} \, ys
\]

**Proof attempt:**

case: \( xs = [ ] \)

\[
\text{length} \, (\, [ ] \, ++ \, ys \, ) \quad \text{(LHS of theorem equation)}
\]

case: \( xs = x:xs' \)

\[
\begin{align*}
[ ] \, ++ \, ys & = ys \\
(x:xs) \, ++ \, ys & = x:(xs \, ++ \, ys)
\end{align*}
\]

\[
\text{length} \, [ ] \, = 0 \\
\text{length} \, (x:xs) \, = \, 1 \, + \, \text{length} \, xs
\]
**Theorem:** For all finite Haskell lists $xs$ and $ys$,

$$\text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys$$

**Proof attempt:**

**case:** $xs = [ ]$

$$\text{length} \; (\; [ ] ++ ys \;) \quad \text{(LHS of theorem equation)}$$

$$= \text{length} \; ( \; ys \;) \quad \text{(unfold ++)}$$

**case:** $xs = x:xs'$

\[
egin{align*}
[ ] ++ ys &= ys \\
(x:xs) ++ ys &= x:(xs ++ ys)
\end{align*}
\]

\[
\begin{align*}
\text{length} \; [ ] &= 0 \\
\text{length} \; (x:xs) &= 1 + \text{length} \; xs
\end{align*}
\]
Theorem: For all finite Haskell lists $xs$ and $ys$,
\[ \text{length}(xs ++ ys) = \text{length } xs + \text{length } ys \]

Proof attempt:

**case:** $xs = []$

\[
\begin{align*}
\text{length } (\ [ \ ] ++ ys) & \quad \text{(LHS of theorem equation)} \\
= \text{length } (ys) & \quad \text{(unfold ++)} \\
= 0 + \text{length } (ys) & \quad \text{(simple arithmetic)} \\
\end{align*}
\]

**case:** $xs = x:xs'$

\[
\begin{align*}
[ ] ++ ys & = ys \\
(x:xs) ++ ys & = x:(xs ++ ys) \\
\text{length } [ ] & = 0 \\
\text{length } (x:xs) & = 1 + \text{length } xs
\end{align*}
\]
**Theorem:** For all finite Haskell lists xs and ys,
length(xs ++ ys) = length xs + length ys

**Proof attempt:**

*case: xs = []*

- length ([ ] ++ ys) (LHS of theorem equation)
- = length (ys) (unfold ++)
- = 0 + length (ys) (simple arithmetic)
- = length [ ] + length (ys) (fold length -- done, we have RHS)

*case: xs = x:xs’*

\[
\begin{align*}
[ ] ++ ys &= ys \\
(x:xs) ++ ys &= x:(xs ++ ys) \\
length [ ] &= 0 \\
length (x:xs) &= 1 + length xs
\end{align*}
\]
**Theorem:** For all finite Haskell lists \( xs \) and \( ys \),

\[
\text{length}(xs ++ ys) = \text{length } xs + \text{length } ys
\]

**Proof attempt:**

case: \( xs = [ ] \)

\[
\begin{align*}
\text{length } ([ ] ++ ys) & \quad \text{(LHS of theorem equation)} \\
= \text{length } ( ys ) & \quad \text{(unfold ++)} \\
= 0 + \text{length } ( ys ) & \quad \text{(simple arithmetic)} \\
= \text{length } [ ] + \text{length } ( ys ) & \quad \text{(fold length)}
\end{align*}
\]

case: \( xs = x:xs' \)
**Theorem:** For all finite Haskell lists `xs` and `ys`,
\[
\text{length}(xs ++ ys) = \text{length} \, xs + \text{length} \, ys
\]

**Proof attempt:**

- **case:** `xs = []`
  
  \[
  \begin{align*}
  \text{length} \, (\text{[]} \, ++ \, ys) & \quad (\text{LHS of theorem equation}) \\
  = \text{length} \, (ys) & \quad (\text{unfold ++}) \\
  = 0 + \text{length} \, (ys) & \quad (\text{simple arithmetic}) \\
  = \text{length} \, \text{[]} + \text{length} \, (ys) & \quad (\text{fold length})
  \end{align*}
  \]

- **case:** `xs = x:xs'`
  
  \[
  \begin{align*}
  \text{length} \, ((x:xs') \, ++ \, ys) & \quad (\text{LHS of theorem equation}) \\
  = \text{length} \, (x:(xs' \, ++ \, ys)) & \quad (\text{unfold ++}) \\
  = 1 + \text{length} \, (xs' \, ++ \, ys)) & \quad (\text{unfold length})
  \end{align*}
  \]

\[
\begin{align*}
\text{[]} \, ++ \, ys & = ys \\
(x:xs) \, ++ \, ys & = x:(xs \, ++ \, ys) \\
\text{length} \, \text{[]} & = 0 \\
\text{length} \, (x:xs) & = 1 + \text{length} \, xs
\end{align*}
\]
**Theorem:** For all finite Haskell lists \(xs\) and \(ys\),

\[
\text{length}(xs ++ ys) = \text{length } xs + \text{length } ys
\]

**Proof attempt:**

- **case:** \(xs = []\)
  
  \[
  \begin{align*}
  \text{length } ( [ ] ++ ys ) &= \text{length } ( ys ) & \text{(LHS of theorem equation)} \\
  &= 0 + \text{length } ( ys ) & \text{(unfold ++)} \\
  &= \text{length } [ ] + \text{length } ( ys ) & \text{(simple arithmetic)} \\
  
  \end{align*}
  \]

- **case:** \(xs = x:xs'\)
  
  \[
  \begin{align*}
  \text{length } ((x:xs') ++ ys) &= \text{length } (x: (xs' ++ ys)) & \text{(LHS of theorem equation)} \\
  &= 1 + \text{length } (xs' ++ ys) & \text{(unfold length)} \\
  
  \end{align*}
  \]

  - **subcase:** \(xs' = []\)
    
    \[
    [ ] ++ ys = ys
    \]
  
  - **subcase:** \(xs' = x':xs''\)
    
    \[
    (x:xs) ++ ys = x:(xs ++ ys)
    \]

- **fold length:** \(\text{length } [ ] = 0\)
- **fold length:** \(\text{length } (x:xs) = 1 + \text{length } xs\)
**Theorem:** For all finite Haskell lists `xs` and `ys`,
\[ \text{length}(\text{xs} ++ \text{ys}) = \text{length} \text{xs} + \text{length} \text{ys} \]

**Proof attempt:**
case: `xs = []`
\[
\begin{align*}
\text{length (} [ ] ++ \text{ys} ) &= \text{length (} \text{ys} ) \\
&= 0 + \text{length (} \text{ys} ) \\
&= \text{length} [ ] + \text{length (} \text{ys} )
\end{align*}
\]
(LHS of theorem equation)
(unfold `++`)
simple arithmetic
(fold `length`)

case: `xs = x:xs'`
\[
\begin{align*}
\text{length ((}x:xs'\text{) ++ }\text{ys}) &= 1 + \text{length (}xs' \text{ ++ }\text{ys}) \\
\text{subcase } xs' &= [ ] \\
&= 1 + \text{length ((}x':xs''\text{) ++ }\text{ys}) \\
\text{subsubcase } xs'' &= [ ] .... \\
\end{align*}
\]
(LHS of theorem equation)
(unfold `++`)
(substitution)
(unfold `length`)

\[
\begin{align*}
[ ] ++ \text{ys} &= \text{ys} \\
(x:xs) ++ \text{ys} &= x:(xs ++ \text{ys}) \\
\text{length } [ ] &= 0 \\
\text{length } (x:xs) &= 1 + \text{length } xs
\end{align*}
\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists $xs$ and $ys$,

$$\text{length}(xs ++ ys) = \text{length } xs + \text{length } ys$$

Proof strategy:

- **Proof by induction on the length of $xs$**
  - must cover both cases: $[ ]$ and $x:xs'$
    - apply *inductive hypothesis* to smaller arguments (smaller lists)
    - In general, Haskell has lots of non-inductive data types like Integers (as opposed to Natural Numbers) so you have to be careful all series of shrinking arguments have base cases
  - use folding/unfolding of Haskell definitions
  - use lemmas/properties you know of basic operations
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists `xs` and `ys`,

\[ \text{length}(xs ++ ys) = \text{length } xs + \text{length } ys \]

Proof: By induction on `xs`.

\[ \text{case } xs = [ ]: \]

- `length [ ] = 0`
- `length (x:xs) = 1 + length xs`
- `xs ++ [ ] = xs2`\[ (++)(x:xs) xs2 = x:(xs ++ xs2)\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,

\[ \text{length}(xs ++ ys) = \text{length } xs + \text{length } ys \]

Proof: By induction on xs.

\[
\begin{align*}
\text{case } xs &= [ ]: \\
\text{length } ([ ] ++ ys) &= (LHS \text{ of theorem})
\end{align*}
\]

- \( \text{length } [ ] = 0 \)
- \( \text{length } (x:xs) = 1 + \text{length } xs \)
- \( (++) [ ] xs2 = xs2 \)
- \( (++) (x:xs) xs2 = x:(xs ++ xs2) \)
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,
\[ \text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys \]

Proof: By induction on xs.

\[
\begin{align*}
\text{case } xs &= [ ]:
\text{length } ([ ] ++ ys) &\quad \text{(LHS of theorem)} \\
= \text{length } ys &\quad \text{(unfold ++)} \\
= 0 + \text{length } ys &\quad \text{(arithmetic)} \\
= \text{length } [ ] + \text{length } ys &\quad \text{(fold length)}
\end{align*}
\]

case done! 

- \text{length } [ ] = 0
- \text{length } (x:xs) = 1 + \text{length } xs
- \text{(++) } [ ] \; xs2 = xs2
- \text{(++) } (x:xs) \; xs2 = x:(xs ++ xs2)
Theorem: For all finite Haskell lists \(xs\) and \(ys\),

\[
\text{length}(xs ++ ys) = \text{length} \hspace{1pt} xs + \text{length} \hspace{1pt} ys
\]

Proof: By induction on \(xs\).

\[
\text{case} \hspace{1pt} xs = x:xs'
\]

| \(\text{length} \hspace{1pt} [\ ]\) | 0 |
| \(\text{length} \hspace{1pt} (x:xs)\) | 1 + \(\text{length} \hspace{1pt} xs\) |

| \((++ \hspace{1pt} [\ ]) \hspace{1pt} xs2\) | \(xs2\) |
| \((++ \hspace{1pt} (x:xs) \hspace{1pt} xs2\) | \(x:(xs ++ xs2)\) |
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,

\[ \text{length}(xs ++ ys) = \text{length } xs + \text{length } ys \]

Proof: By induction on xs.

\[ \text{case } xs = x:xs' \]

IH: \[ \text{length } (xs' ++ ys) = \text{length } xs' + \text{length } ys \]

- \[ \text{length } [ ] = 0 \]
- \[ \text{length } (x:xs) = 1 + \text{length } xs \]

\[ (+) [ ] xs2 = xs2 \]
\[ (+) (x:xs) xs2 = x:(xs ++ xs2) \]
Theorem: For all finite Haskell lists \(xs\) and \(ys\),

\[
\text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys
\]

Proof: By induction on \(xs\).

case \(xs = x:xs'\)

IH: \(\text{length} \; (xs' ++ ys) = \text{length} \; xs' + \text{length} \; ys\)

\[
\text{length} \; ((x:xs') ++ ys) \quad \text{(LHS of theorem)}
\]

\[
\begin{align*}
\text{length} \; [ ] &= 0 \\
\text{length} \; (x:xs) &= 1 + \text{length} \; xs \\
(++) \; [ \; ] \; xs2 &= xs2 \\
(++) \; (x:xs) \; xs2 &= x:(xs ++ xs2)
\end{align*}
\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,

\[ \text{length}(xs ++ ys) = \text{length} \, xs + \text{length} \, ys \]

Proof: By induction on xs.

\[
\begin{align*}
\text{case } xs &= x : xs' \\
\text{IH: } \text{length} \,(xs' ++ ys) &= \text{length} \, xs' + \text{length} \, ys
\end{align*}
\]

\[
\begin{align*}
\text{length} \,(x : (xs' ++ ys)) &= \text{length} \,(x : (xs' ++ ys)) \\
&= 1 + \text{length} \, xs & \text{(unfold ++)}
\end{align*}
\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,
\[ \text{length}(\text{xs} ++ \text{ys}) = \text{length} \text{xs} + \text{length} \text{ys} \]

Proof: By induction on xs.

case \( \text{xs} = \text{x}:\text{xs}' \)

IH: \[ \text{length} (\text{xs}' ++ \text{ys}) = \text{length} \text{xs}' + \text{length} \text{ys} \]

\[
\begin{align*}
\text{length}((\text{x}:\text{xs}') ++ \text{ys}) & \quad \text{(LHS of theorem)} \\
= \text{length} (\text{x} : (\text{xs}' ++ \text{ys})) & \quad \text{(unfold ++)} \\
= 1 + \text{length} (\text{xs}' ++ \text{ys}) & \quad \text{(unfold length)}
\end{align*}
\]

\[
\begin{align*}
\text{length} [ ] & = 0 \\
\text{length} (\text{x}:\text{xs}) & = 1 + \text{length} \text{xs} \\
(++) [ ] \text{xs2} & = \text{xs2} \\
(++) (\text{x}:\text{xs}) \text{xs2} & = \text{x}: (\text{xs} ++ \text{xs2})
\end{align*}
\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,

\[ \text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys \]

Proof: By induction on xs.

case xs = x:xs’
   IH: \text{length} \; (xs’ ++ ys) = \text{length} \; xs’ + \text{length} \; ys

\[
\begin{align*}
\text{length} \; ((x:xs’) ++ ys) & \quad \text{(LHS of theorem)} \\
= \text{length} \; (x : (xs’ ++ ys)) & \quad \text{(unfold ++)} \\
= 1 + \text{length} \; (xs’ ++ ys) & \quad \text{(unfold length)} \\
= 1 + (\text{length} \; xs’ + \text{length} \; ys) & \quad \text{(by IH)}
\end{align*}
\]

\[
\begin{align*}
\text{length} \; [ ] &= 0 \\
\text{length} \; (x:xs) &= 1 + \text{length} \; xs \\
(x:xs) ++ [ ] &= xs \\
(x:xs) ++ xs2 &= x:(xs ++ xs2)
\end{align*}
\]
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists xs and ys,

\[
\text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys
\]

Proof: By induction on xs.

\[
\begin{align*}
\text{case } xs = & \; x:xs' \\
\text{IH: } \text{length} \; (xs' ++ ys) = \text{length} \; xs' + \text{length} \; ys \\
\text{length} \; ((x:xs') ++ ys) & = \text{length} \; (x : (xs' ++ ys)) \\
& = 1 + \text{length} \; (xs' ++ ys) \\
& = 1 + (\text{length} \; xs' + \text{length} \; ys) \\
& = \text{length} \; (x:xs') + \text{length} \; ys
\end{align*}
\]

\[
\begin{align*}
(++) \; [ ] \; xs2 & = xs2 \\
(++) \; (x:xs) \; xs2 & = x:(xs ++ xs2)
\end{align*}
\]

length [ ] = 0
length (x:xs) = 1 + length xs
Proofs over Recursive Haskell Functions

Theorem: For all finite Haskell lists \( xs \) and \( ys \),
\[
\text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys
\]

Proof: By induction on \( xs \).

\[
\text{case } xs = x : xs' \\
\text{IH: } \text{length} \; (xs' ++ ys) = \text{length} \; xs' + \text{length} \; ys
\]

\[
\begin{align*}
\text{length} \; ((x : xs') ++ ys) & \quad \text{(LHS of theorem)} \\
= \text{length} \; (x : (xs' ++ ys)) & \quad \text{(unfold ++)} \\
= 1 + \text{length} \; (xs' ++ ys) & \quad \text{(unfold length)} \\
= 1 + (\text{length} \; xs' + \text{length} \; ys) & \quad \text{(by IH)} \\
= \text{length} \; (x : xs') + \text{length} \; ys & \quad \text{(reparenthesizing and folding length we have RHS with } x : xs' \text{ for } xs)
\end{align*}
\]

case done!

All cases covered! Proof done!

\[
\begin{align*}
\text{length} \; [ ] &= 0 \\
\text{length} \; (x : xs) &= 1 + \text{length} \; xs \\
(+) \; [ ] \; xs2 &= xs2 \\
(+) \; (x : xs) \; xs2 &= x : (xs \; (+) \; xs2)
\end{align*}
\]
Exercises

To test your understanding, try to prove the following:

**Theorem 1:** for all finite lists $xs$, $ys$. \(\text{listSum}(xs ++ ys) = \text{listSum} \; xs + \text{listSum} \; ys\)

\[
drop \; n \; \; \; [\; \; \;] = \; [\; \; \;]
\]
\[
drop \; n \; (x:xs) = \begin{cases} 
\text{if } n \leq 0 \text{ then } x:xs \\
\text{else } \; \text{drop} \; (n-1) \; xs
\end{cases}
\]

**Theorem 2:** for all finite lists $xs$, natural numbers $n$ and $m$,
\[
\text{drop} \; n \; (\text{drop} \; m \; xs) = \text{drop} \; (n+m) \; xs
\]

**Hint:** split the inductive case where $xs = x:xs$ into 3 subcases:

Case $xs = x:xs$:

- subcase $m = 0$ and $n = 0$: ...
- subcase $m = 0$ and $n = n' + 1$ for some natural number $n'$ (ie: $n > 0$): ...
- subcase $m = m' + 1$ for some natural number $m'$ (ie: $m > 0$): ...
Summary

• Haskell is
  – a functional language emphasizing immutable data
  – where every expression has a type:
    • Char, Int, (Char, Int, Float), [ Int ], [ [ [ (Char, [ [ Int ] ] ) ] ] ]
    • Char -> Int, (Char, Char) -> Int -> [ (Char, Int) ]
    • String = [ Char ]

• Reasoning about Haskell programs involves
  – substitution of “equals for equals,” unlike in Java or C
  – mathematical calculation:
    • unfold function abstractions
    • push symbolic names around like we do in mathematical proofs
    • reason locally using properties of operations (eg: + commutes)
    • use induction hypothesis
    • fold function abstractions back up

• Homework: Install Haskell. Read LYAHFGG Intro, Chapter 1